摘要

serverless近些年应用广泛，原因是他的服务粒度小，在节省资源的同时可以减少很大一部分服务支出，对服务提供商和用户来说是双赢的。然而虽然serverless很灵活，可以在高并发情形下表现的很好，但是“冷启动问题”却总会消耗很长一段时间，这段时间除了做initialization以外，没有做任何有价值的工作。
本文提出了一种基于池的解决方式，比常规的冷启动方法快了85%。

1. 简介

1.1 无服务平台

目前主流的serverless平台主要还是为用户提供一个简单易开发的模型。

1.2 KKS

主要提供三个部分：

- 为容器镜像建立一个端到端的源到URL的部署工具
- 在容器之间进行的事件管理和事件传送
- 提供一个请求驱动的自动缩放模块，该模块可以缩小到0（即产生和serverless同等的按使用收费模式）
- 利用服务网格网络框架IStio来解决路由问题

这些组件使得KKS能够满足serverless框架的需求

其工作原理基本上是基于K8S的，每个service都有自动缩放和负载均衡的功能，同时有自己一个特俗的路由id。另外每个版本都是独一无二的，有自己相应的标识符

1.3 冷启动问题

初始化的时间对每个function来说都是不可避免的，与此同时也会带来相应的延迟，这在serverless平台上非常常见。

provision 配置设置

而在KS中可以将延迟分为两部分：

- 平台延迟：对网络的初始化，网络的注入、以及Pod的配置
- 应用初始化延迟：应用的包的导入、一些依赖的配置

比如在我们的实验中，一个HTTP应用的初始化需要5s，而一个tensorflow的则需要40s

我们提出的解决方案是用一个资源池存放预热好的容器组，这些容器组可以被l立刻获得，从而消除了部分冷启动的代价。

2. 实现

2.1 容器组的池

revision 修订

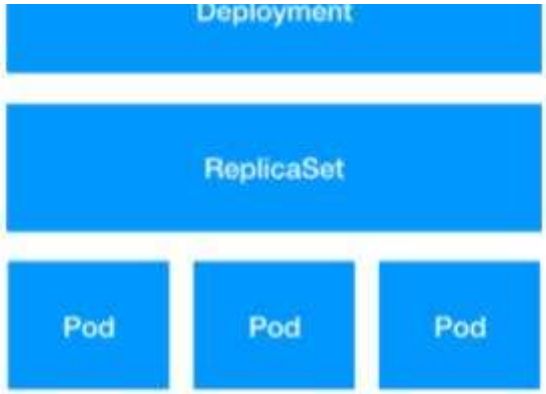


Figure 2: Knative Revision Component Structure

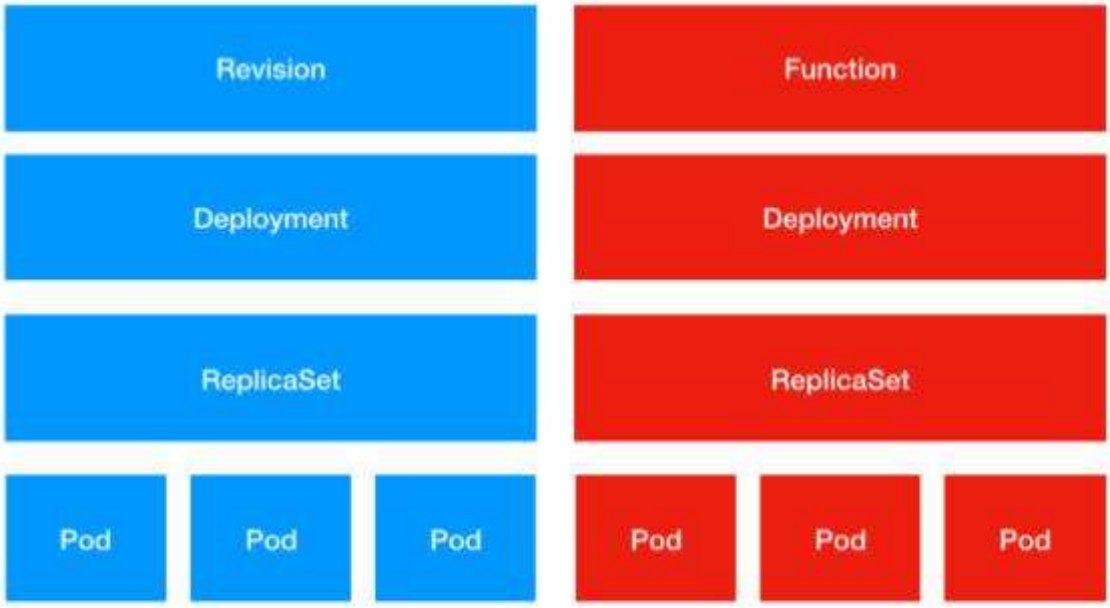


Figure 3: Our Pool Structure

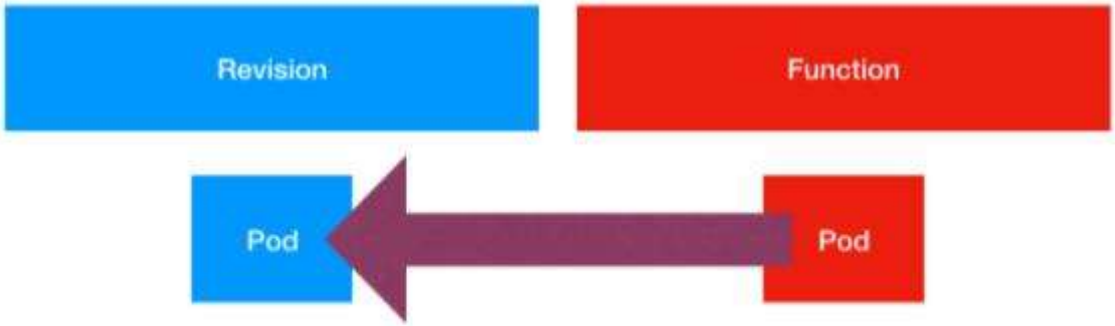


Figure 4: Migration of pods in the pool

2.2 Pod迁移

Knative的缩放器原理

用的是reversion用的数据结构

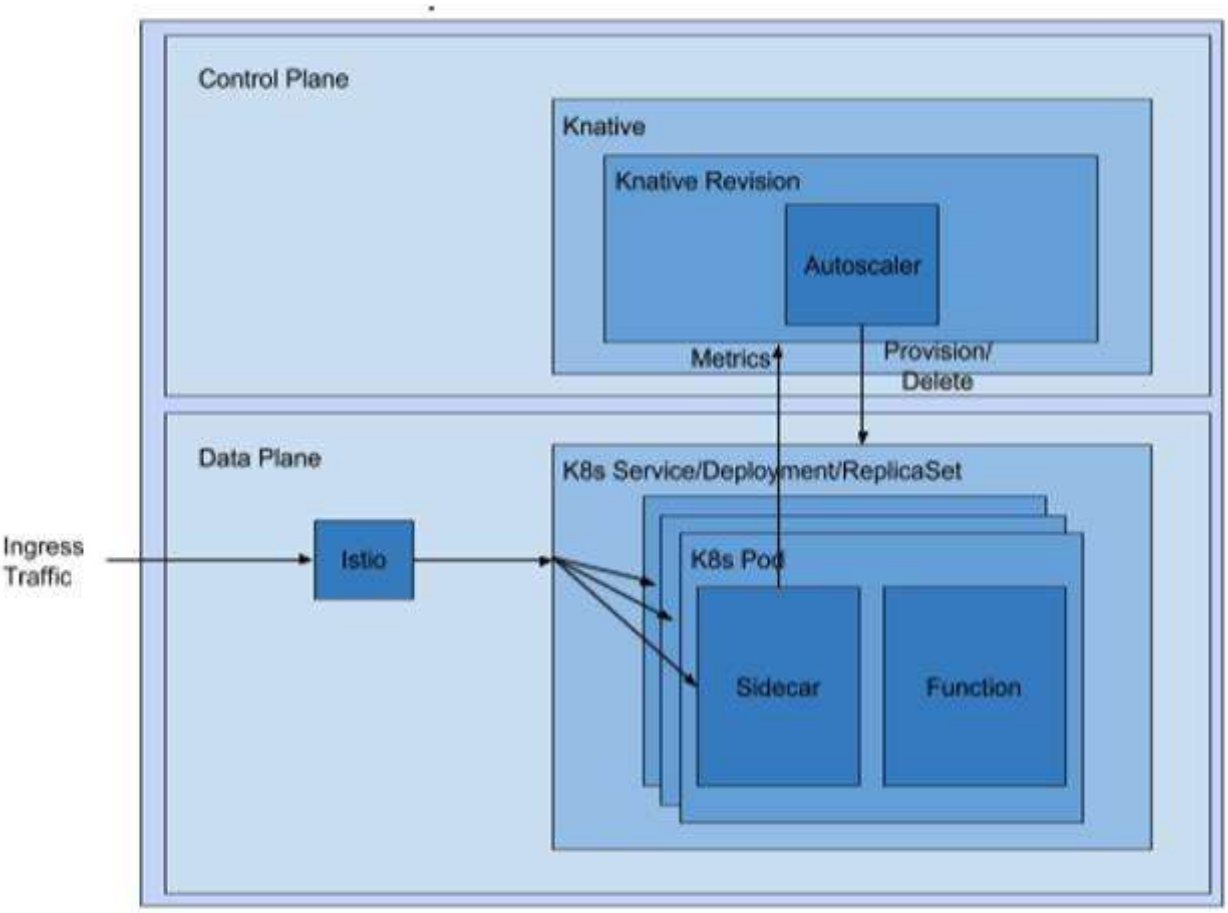
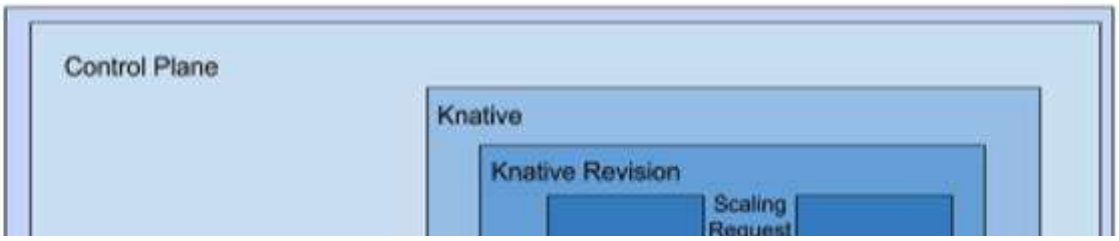


Figure 5: Original Autoscaling Logic

利用Pod中的sidecar容器来收集流量、拥塞信息反应给自动缩放器，自动所缩放器就会通过deployment中pod的设置数量进行调整

改进



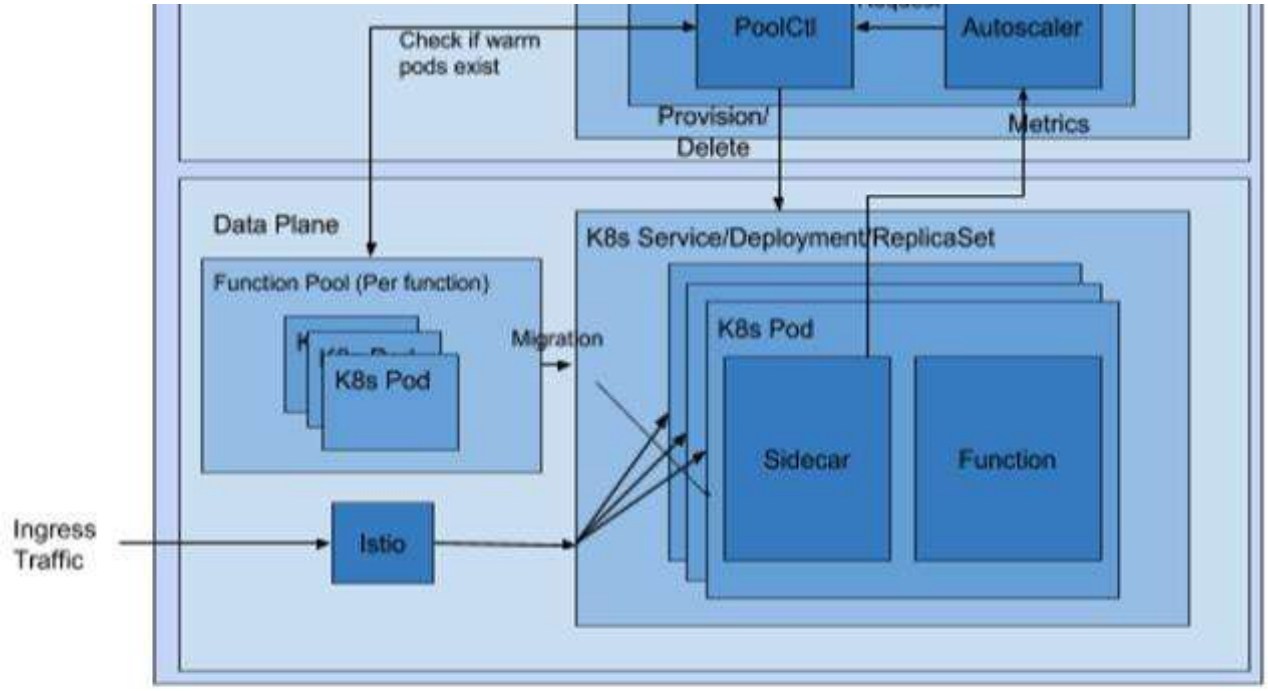


Figure 6: Our Autoscaling (with pool) Logic

在改变repilcaSet中的pod数量之前，先检测一下pool中的pod数量，如果够的话，就从pool中直接迁移给replicaSet，否则还是按照传统的机制，重新生成处新的pod出来。

3. 评估

冷启动节省大概一半多的时间

资源池里面的pods多了反而不利于提高加速时间